IN THE CLAIMS

Please amend the claims as indicated in the rewritten claims listed below:

Claim Amendments:

Claim 1 (Canceled)

Claim 2 (Currently Amended): A method for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in a memory~~, the memory being~~ that is organized into separately addressable pages of memory~~memory blocks~~, the method comprising:

executing a program, the program having a series of computer-executable instructions executed by the at least one processor, the at least one processor residing in a system hardware platform;

verifying that the program is valid using a verification engine including verification instructions for execution on the at least one processor residing in the system hardware platform, the program being valid when the program does not include the unauthorized code, wherein the verification engine resides in a system software layer, the system software layer performing hardware-interface and resource-allocating functions;

ensuring that the program is not executed without also dynamically performing the verifying~~, the verifying based on a next instruction to be executed in the series of computer-executable instructions in the program~~; and

continuing execution of the program after dynamically performing the verifying by executing the next instruction as long as the verifying determines that the program is valid and generating a protective response when the verifying does not determine ~~determines~~ that the program is ~~not~~ valid;

wherein the verifying that the program is valid comprises executing the verification instructions of the verification engine, the verification instructions causing the at least one processor to perform a method comprising:

identifying the next instruction to be executed in the program;

for the next instruction to be executed in the program, determining an identifying value for a ~~memory block~~ page of memory that contains the next instruction;

determining whether the identifying value satisfies a validation condition, wherein the determining as to whether the identifying value satisfies the validation condition ~~requires~~ comprises comparing the identifying value of the ~~memory block~~ page of memory with a set of reference values; and

determining that the program is valid only when the validation condition is satisfied.

Claim 3 (Currently Amended): The method of claim 2, wherein the validation condition is that the identifying value of the ~~memory block~~ page of memory matches any reference value in the set of reference values.

Claim 4 (Currently Amended): The method of claim 2, wherein the validation condition is that the identifying value of the ~~memory block~~ page of memory differs from each reference value in the set of reference values.

Claim 5 (Canceled)

Claim 6 (Currently Amended): A method for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in memory~~, the memory being~~ that is organized into separately addressable ~~memory blocks~~ pages of memory, the method comprising:

executing a program, the program having a series of computer-executable instructions;

verifying that the program is valid as the program executes, the program being valid when the program does not include the unauthorized code, the verifying based on a current instruction to be executed when executing the series of instructions in the program;

generating a protective response when the verifying determines that the program is not valid; and

executing the current instruction when the verifying determines that the program is valid;

wherein the verifying that the program is valid comprises:

identifying the current instruction to be executed when executing the series of instructions, the current instruction being one of the series of instructions being executed identified for submission to the processor for execution and not yet executed at a time of the identifying;

for at least the current instruction, computing a hash value as a function of a subset of contents of a current ~~memory block~~ page of memory that contains the current instruction;

determining, during the executing of the series of instructions, whether the hash value satisfies a validation condition by comparing the hash value of the current ~~memory block~~ page of memory with a set of reference values; and

determining that the program is valid when the hash value satisfies the validation condition, and determining that the program is not valid when the hash value does not satisfy the validation condition;

wherein the computing of the hash value comprises applying a mask to the current ~~memory block~~ page of memory, the mask being a data structure that designates at least one byte of the current ~~memory block~~ page of memory to be ignored in the computing of the hash value, the data structure designating less than an entire ~~memory block~~ page of memory so that the hash value is based on only part of the contents of the current ~~memory block~~ page of memory.

Claim 7 (Currently Amended): The method of claim 6, further comprising:

identifying, an indeterminate portion of the current ~~memory block~~ page of memory, the indeterminate portion being non-indicative of validity of the current ~~memory block~~ page of memory as a whole; and

configuring the mask so that the mask designates at least the indeterminate portion to be ignored when generating the hash value.

Claim 8 (Currently Amended): The method of claim 2, wherein the verifying that the program is valid further comprises:

for each of the separately addressable ~~memory blocks~~ pages of memory, indicating in a structure whether the ~~memory block~~ page is valid;

accessing the structure to determine whether the ~~memory block~~ page of memory is valid prior to the determining of the identifying value; and

performing the determining of the identifying value when the structure does not indicate that the ~~memory block~~ page of memory is valid and directly allowing execution of the next instruction when the structure indicates that the ~~memory block~~ page of memory is valid.

Claim 9 (Currently Amended): The method of claim 8, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the plurality of ~~memory blocks~~ pages of memory are ~~is~~ validated comprises setting one of the hardware attribute indicators, the one hardware attribute indicator corresponding to the ~~memory block~~ page of memory.

Claim 10 (Previously Presented): The method of claim 9, in which the hardware attribute indicators are execute and write permission attributes associated with an entry in a translation lookaside buffer.

Claim 11 (Currently Amended): The method of claim 8, wherein the structure comprises a software data structure, and wherein the indicating in the structure whether the plurality of ~~memory blocks~~ pages of memory are ~~is~~ validated comprises making a corresponding entry in the software data structure.

Claim 12 (Currently Amended): The method of claim 8, wherein the determining of the identifying value for the ~~memory block~~page of memory and the determining of whether the identifying value satisfies the validation condition are performed only when the structure does not indicate that the ~~memory block~~page of memory is valid, the method further comprising:

when the structure does not indicate that the ~~memory block~~page of memory is valid, modifying the structure to indicate that the ~~memory block~~page of memory is valid when the identifying value satisfies the validation condition.

Claim 13 (Currently Amended): The method of claim 12, further comprising:

sensing modification of one of the ~~memory blocks~~pages of memory that the structure indicates is valid and, in response to the modification, setting its indication in the structure to indicate that the ~~memory block~~page of memory is not valid.

Claim 14 (Currently Amended): The method of claim 8, wherein the verifying that the program is valid further comprises:

determining a branch history for the next instruction; and

checking whether the ~~memory blocks~~pages of memory in which instructions in the branch history are located are valid, the validation condition including the requirement that each checked ~~memory block~~ in the branch history is valid.

Claim 15 (Previously Presented): The method of claim 2, wherein the determining of the identifying value and the determining as to whether the validation condition has been satisfied are performed only after a triggering event occurs.

Claim 16 (Previously Presented): The method of claim 15, wherein the triggering event is writing of at least one new unit of code or data to any physical component within the computer.

Claim 17 (Currently Amended): The method of claim 15, in which the triggering event is an attempted execution of any instruction located on any unverified ~~memory block~~page of memory.

Claim 18 (Currently Amended): The method of claim 15, wherein the triggering event is an attempted execution of any instruction located on any unverified ~~memory block~~page of memory of newly installed software.

Claim 19 (Previously Presented): The method of claim 15, further comprising triggering the verification of the computer instructions depending on an identity of a user of the computer, the user having caused the next instruction to be identified for execution.

Claim 20 (Canceled)

Claim 21 (Previously Presented): The method of claim 15, further comprising triggering dynamic verification depending on a context in which the next instruction is submitted for execution, wherein the context is a level of security clearance associated with the computer, a user of the computer, or a program of which the next instruction is a part.

Claim 22 (Previously Presented): The method of claim 2, wherein the identifying of the next instruction is performed for only a sample of the series of instructions.

Claim 23 (Previously Presented): The method of claim 22, wherein the sample is a time-sampled subset of the series of instructions.

Claim 24 (Previously Presented): The method of claim 22, wherein the sample is a sequentially sampled subset of the series of instructions.

Claim 25 (Currently Amended): The method of claim 22, wherein the sample is a subset of the series of instructions sampled spatially, the sampling being over a range of ~~memory block~~page of memory identifiers.

Claim 26 (Currently Amended): The method of claim 2, wherein the protective response comprises termination of a software entity with which the current ~~memory block~~page of memory is associated.

Claim 27 (Currently Amended): The method of claim 2, wherein the protective response comprises suspension of execution of a software entity with which the current ~~memory block~~page of memory is associated.

Claim 28 (Previously Presented): The method of claim 2, wherein the protective response comprises a message posted to a user, system administrator, or other predetermined recipient.

Claim 29 (Previously Presented): The method of claim 2, wherein:

the computer includes a virtual machine running in a direct execution mode on an underlying hardware platform via an intermediate software layer; and

the protective response comprises a switching of an execution mode of the virtual machine from the direct execution mode to a binary translation mode.

Claim 30 (Currently Amended): The method of claim 2, wherein:

the computer includes a virtual machine running on ~~an~~ the underlying hardware platform via an intermediate software layer; and

the protective response includes checkpointing the state of the virtual machine.

Claim 31 (Currently Amended): The method of claim 2, wherein the protective response is a first possible response, the method further comprising:

associating the first possible response with the ~~memory block~~page of memory;

associating a second possible response with a different ~~memory block~~page of memory; and

upon detection of failure of the next instruction to satisfy the validation condition, identifying which one of the possible responses is associated with the ~~memory block~~page of memory, and generating the one possible response associated with the ~~memory block~~page of memory in which the next instruction is located.

Claim 32 (Currently Amended): The method of claim 2, further comprising:

associating reference values from the set of reference values with respective programs such that each association signifies that the reference value corresponds to a ~~memory block~~page of memory storing instructions for one of the programs; and

tracking which of the respective programs is being executed and the association between the matching reference value and the corresponding one of the programs.

Claim 33 (Currently Amended): The method of claim 2, wherein:

the computer includes a virtual machine (VM) running on ~~an~~ the underlying hardware platform via ~~an intermediate~~ the system software layer operable to switch the virtual machine between a direct execution mode and a binary translation mode; and

the series of instructions comprise VM-issued instructions issued in conjunction with binary translation of any of the VM-issued instructions, the VM-issued instructions thereby being verified.

Claim 34 (Canceled)

Claim 35 (Currently Amended): A non-transitory machine readable storage medium embodying executable code for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in a memory~~, the memory being~~ that is organized into separately addressable ~~memory blocks~~pages of memory, the executable code being a verification engine causing the computer to perform a method having operations of:

executing a program, the program having a series of computer-executable instructions executed by the at least one processor, the at least one processor residing in a system hardware platform;

verifying that the program is valid using a verification engine including verification instructions for execution on the at least on processor residing in the system hardware platform, the program being valid when the program does not include the unauthorized code, wherein the verification engine resides in a system software layer, the system software layer performing hardware-interface and resource-allocating functions;

ensuring that the program is not executed without also dynamically performing the verifying, the verifying based on a next instruction to be executed in the series of computer-executable instructions in the program; and

continuing execution of the program ~~after dynamically performing the verifying by executing the next instruction~~ as long as the verifying determines that the program is valid and generating a protective response when the verifying does not determine~~determines~~ that the program is ~~not~~ valid;

wherein the verifying that the program is valid comprises executing the verification instructions of the verification engine, the verification instructions causing the at least one processor to perform a method comprising:

identifying the next instruction of the series of computer-executable instructions to be executed when executing the program;

for the next instruction, and during the execution of the program, determining an identifying value for a ~~memory block~~page of memory that contains the next instruction;

determining whether the identifying value satisfies a validation condition, wherein the determining as to whether the identifying value satisfies the validation condition ~~requires~~ comprises comparing the identifying value of the ~~memory block~~page of memory with a set of reference values; and

determining that the program is valid only when the validation condition is satisfied.

Claim 36 (Currently Amended): The machine readable storage medium of claim 35, wherein the validation condition is that the identifying value of the ~~memory block~~page of memory matches any reference value in the set of reference values.

Claim 37 (Currently Amended): The machine readable storage medium of claim 35, wherein the validation condition is that the identifying value of the ~~memory block~~page of memory differs from each reference value in the set of reference values.

Claim 38 (Currently Amended): The machine readable storage medium of claim 35, wherein: the identifying value is a hash value that is computed as a function of contents of the ~~memory block~~page of memory.

Claim 39 (Currently Amended): The machine readable storage medium of claim 38, wherein some of the contents of the ~~memory block~~page of memory are ignored when computing the hash value, the ignored portion being defined by a subset selection structure.

Claim 40 (Previously Presented): The machine readable storage medium of claim 39, wherein the subset selection structure is a mask.

Claim 41 (Currently Amended): The machine readable storage medium of claim 35, wherein the verifying that the program is valid further comprises:

for each of the separately addressable ~~memory blocks~~pages of memory, indicating in a structure whether the ~~memory block~~ is valid;

accessing the structure to determine whether the ~~memory block~~page of memory is valid prior to the determining of the identifying value; and

performing the determining of the identifying value when the structure does not indicate that the ~~memory block~~page of memory is valid and directly allowing execution of the next instruction when the structure indicates that the ~~memory block~~page of memory is valid.

Claim 42 (Currently Amended): The machine readable storage medium of claim 41, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the plurality of ~~memory blocks~~pages of memory is valid comprises setting one of the hardware attribute indicators corresponding to the ~~memory block~~page of memory.

Claim 43 (Previously Presented): The machine readable storage medium of claim 42, in which the hardware attribute indicators are execute and write permission attributes associated with an entry in a translation lookaside buffer.

Claim 44 (Previously Presented): The machine readable storage medium of claim 41, wherein the structure comprises a software data structure, and wherein the verifying further comprises selecting for verification only a sample of the next instructions.

Claim 45 (Canceled)

Claim 46 (Currently Amended): The machine readable storage medium of claim 35, wherein:

the verification engine resides in an intermediate virtualization layer in the system software layer between a virtual machine and ~~a~~ the hardware platform of the computer;

the next instruction is issued by the virtual machine; and

the verifying that the program is valid is performed while copying or translating in conjunction with binary translation of instructions for the virtual machine.

Claim 47 (Previously Presented): The machine readable storage medium of claim 46, wherein the protective response comprises switching the intermediate virtualization layer to a binary translation mode.

Claim 48 (Currently Amended): The method of claim 2, wherein the identifying value is a hash value that is computed as a function of all contents of the ~~memory block~~page of memory.

Claim 49 (Currently Amended): The method of claim 2, wherein the identifying value is a hash value that is computed as a function of contents of the ~~memory block~~page of memory such that some of the contents of the ~~memory block~~page of memory are ignored when computing the hash value, the ignored portion being defined by a subset selection structure.

Claim 50 (Currently Amended): A method for verifying the validity of instructions in a computer that includes at least one physical processor that executes instructions stored in a memory of the computer, the memory being organized into separately addressable ~~memory blocks~~pages of memory, the computer also including system software and verification software residing in a system software layer, the verification software including verification instructions for execution on the at least one physical processor residing in a system hardware platform~~, the system software performing hardware interface, resource allocating, and control functions of an operating system~~, the method comprising:

monitoring instructions to be executed by the at least one physical processor under control of the system software layer, the system software layer performing hardware-interface and resource-allocating functions;

as long as the instructions to be executed under control of the system software are stored in one or more ~~memory blocks~~pages of memory for which validation is deemed unnecessary, allowing the instructions to be executed under control of the system software on the at least one physical processor; and

detecting that an unvalidated instruction is to be executed under control of the system software, the unvalidated instruction being stored in a ~~memory block~~page of memory that has not been validated, and, before allowing the unvalidated instruction to execute, attempting to validate the unvalidated ~~memory block~~ by executing the verification instructions of the verification software in the system software layer, the verification instructions causing the at least one physical processor to perform a method comprising~~by performing the following steps~~:

determining an identifying value for the unvalidated ~~memory block~~page of memory;

comparing the identifying value of the unvalidated ~~memory block~~page of memory with a set of reference values;

if the identifying value satisfies a validation condition, allowing execution after satisfying the validation condition, under control of the system software layer, of instructions stored in the unvalidated ~~memory block~~page of memory; and

if the identifying value does not satisfy the validation condition, generating a response.

Claim 51 (Currently Amended): The method of claim 50, wherein the validation condition is that the identifying value of the ~~memory block~~page of memory matches any reference value in the set of reference values.

Claim 52 (Currently Amended): The method of claim 50, wherein the validation condition is that the identifying value of the ~~memory block~~page of memory differs from each reference value in the set of reference values.

Claim 53 (Currently Amended): The method of claim 50, wherein the determining of the identifying value comprises computing a hash value as a function of contents of the unvalidated ~~memory block~~page of memory, the identifying value being the hash value.

Claim 54 (Currently Amended): The method of claim 53, wherein some of the contents of the unvalidated ~~memory block~~page of memory are ignored when computing the hash value, the ignored portion being defined by a subset selection structure.

Claim 55 (Previously Presented): The method of claim 54, wherein the subset selection structure is a mask.

Claim 56 (Currently Amended): The method of claim 50, wherein the detecting that an unvalidated instruction is to be executed comprises:

for each of the separately addressable ~~memory blocks~~pages of memory, indicating in a structure whether the ~~memory block~~page of memory is valid;

accessing the structure to determine whether one of the ~~memory blocks~~pages of memory containing the instructions to be executed is valid prior to the determining of the identifying value; and

detecting that the instruction to be executed is unvalidated when the structure does not indicate that the ~~memory block~~page of memory that contains the instruction to be executed is valid.

Claim 57 (Currently Amended): The method of claim 56, wherein the structure comprises a group of hardware attribute indicators, and wherein the indicating in the structure whether the ~~memory block~~page of memory is valid comprises setting one of the hardware attribute indicators corresponding to the ~~memory block~~page of memory.

Claim 58 (Previously Presented): The method of claim 56, wherein the structure comprises a software data structure.

Claim 59 (Currently Amended): The method of claim 50, wherein:

the verification software resides in an intermediate virtualization layer of the system software layer between a virtual machine and a hardware platform of the computer;

the instructions to be executed that are being monitored comprise instructions issued by the virtual machine; and

the verifying of the validity of the instructions is carried out while copying or translating in conjunction with binary translation of the instructions to be executed.

Claim 60 (Currently Amended): The method of claim 50, wherein the detecting that an unvalidated instruction is to be executed further comprises:

determining a branch history for the unvalidated instruction; and

checking whether ~~memory blocks~~pages of memory in which instructions in the branch history are located are valid, the validation condition including the requirement that each checked ~~memory block~~page of memory in the branch history is valid.

Claim 61 (Previously Presented): The method of claim 50, wherein monitoring of the instructions to be executed and the detecting that the unvalidated instruction is to be executed are performed only after a triggering event occurs.

Claim 62 (Currently Amended): The method of claim 2, wherein ensuring that the program is not executed without dynamically performing the verifying comprises interrupting execution of the series of computer-executable instructions of the program while dynamically performing the verifying to execute the verification instructions of the verification engine using the at least one processor to perform.

Claim 63 (New): A method for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in a memory, the memory being organized into separately addressable pages of memory, the method comprising:

setting execute permission indicators for the separately addressable pages of memory in a structure, wherein an execute permission indicator for a page of memory indicates whether the page of memory is valid;

executing a program, the program having a series of computer-executable instructions being executed by the at least one processor;

determining when the program attempts to execute an instruction on a page of memory in which an indicator for the page of memory does not indicate that the page of memory is valid;

dynamically verifying that the page of memory is valid, the page of memory being valid when the page of memory does not include unauthorized code; and

continuing execution of the program after dynamically performing the verifying by executing the instruction as long as the verifying determines that the page of memory is valid and generating a protective response when the verifying determines that the program is not valid.

Claim 64 (New): The method of claim 63, wherein the structure stores the execute permission indicators in a hardware structure, the method further comprising inspecting the hardware data structure to determine if the execute permission indicator indicates whether the page of memory is valid.

Claim 65 (New): The method of claim 64, further comprising receiving a fault generated by the at least one processor, the fault being generated when the program attempts to execute the instruction on the page of memory in which the indicator for the page of memory does not indicate the page of memory is valid,

wherein the verifying is performed upon receiving the fault.

Claim 66 (New): The method of claim 63, further comprising:

setting access control indicators for pages of memory marked as valid by the execute permission indicators;

determining a modification to a page of memory marked as valid, the modification violating the access control indicator for the page of memory; and

changing the execute permission indicator for the modified page of memory to indicate that the modified page of memory is not valid in response to determining the modification.

Claim 67 (New): The method of claim 66, wherein the access control indicator indicates the page of memory is read-execute only.

Claim 68 (New): The method of claim 63, wherein the structure comprises a software data structure, the method further comprising inspecting the software data structure to determine if the execute permission indicator indicates whether the page of memory is valid.

Claim 69 (New): The method of claim 63, wherein the computer includes a virtual machine running in a direct execution mode on an underlying hardware platform via an intermediate software layer.

Claim 70 (New): A method for protecting a computer from unauthorized code, the computer including at least one processor that executes instructions stored in a memory, the memory being organized into separately addressable pages of memory, the method comprising:

during execution of a program having a series of computer-executable instructions being executed by the at least one processor, determining one or more pages of memory from the separately addressable pages of memory to translate;

verifying, before the one or more pages of memory are stored in a translation lookaside buffer, that the one or more pages of memory are valid, the one or more pages of memory being valid when the one or more pages of memory do not include unauthorized code;

storing the one or more pages of memory in a translation lookaside buffer;

determining an instruction for execution;

determining if the instruction is found in a page of memory stored in the translation lookaside buffer; and

if the instruction is found in a page of memory stored in the translation lookaside buffer, executing the instruction without verifying of the page of memory.

Claim 71 (New): The method of claim 70, further comprising setting one or more access control indicators for the one or more pages of memory stored in the translation lookaside buffer.

Claim 72 (New): The method of claim 71, wherein the one or more access control indicators indicate the one or more pages are read-execute only.

Claim 73 (New): The method of claim 72, wherein upon a write to a page of memory of the one or more pages of memory, the method further comprising:

removing the page of memory from the translation lookaside buffer, wherein verifying is performed before the page of memory is re-stored in the translation lookaside buffer.

Claim 74 (New): The method of claim 70, wherein:

the computer includes a virtual machine (VM) running on an underlying hardware platform via an intermediate software layer; and

the series of instructions comprise VM-issued instructions issued in conjunction with binary translation of any of the VM-issued instructions, wherein it is determined if the VM-issued instructions are found in a page of memory stored in the translation lookaside buffer.

Claim 75 (New): The method of claim 2, wherein:

the series of computer-executable instructions belong to an instruction set associated with the at least one processor, and

the verification instructions belong to the instruction set associated with the at least one processor.

Claim 76 (New): The method of claim 2, wherein the dynamically performing of the verifying is initiated from the system software layer.